공학석사학위논문

플래시 저장장치의 성능 및 신뢰성 평가를 위한 무순서 플래시 메모리 제어기의 고해상 실시간 시뮬레이션

High-fidelity and Real-time Simulation of Out-of-Order Flash Memory Controller for Performance and Reliability Testing of Flash-based Storage Device

2015 년 2 월

서울대학교 대학원

전기·컴퓨터공학부

배 종 보

플래시 저장장치의 성능 및 신뢰성 평가를 위한 무순서 플래시 메모리 제어기의 고해상 실시간 시뮬레이션

High-fidelity and Real-time Simulation of Out-of-Order Flash Memory Controller for Performance and Reliability Testing of Flash-based Storage Device

지도교수 민 상 렬

이 논문을 공학석사학위논문으로 제출함

2015 년 2 월

서울대학교 대학원

전기·컴퓨터공학부

배 종 보

배종보의 석사학위논문을 인준함

2015 년 2 월

위 원 장 :	신 현 식	(인)
부위원장 :	민 상 렬	(인)
위 원 :	하 순 회	(인)
위 원 :	김 지 홍	(인)

초록

플래시 메모리 기반 저장장치의 개발은 저장장치 하드웨어와 내부 소프트웨어의 설계가 병행되어 진행된다. 소프트웨어의 개발시 변화에 따른 실제 저장장치의 성능변화나 신뢰성 변화를 확인하는데 있어 목표 저장장치의 시제품 제작을 통한 성능평가와 신뢰성 테스트는 한계가 있어 이를 위한 소프트웨어 환경이 필요하다.

본 논문에서는 정확한 시간지연과 오류상황을 모사하는 무순서 플래시 메모리 제어기의 시뮬레이션을 제시한다. 무순서 플래시 메모리 제어기는 다수의 플래시 연산들을 도착 순서에 관계없이 실행하여 플래시 저장장치의 성능을 향상시키는 플래시 메모리 제어기로, 이 무순서 플래시 메모리 제어기의 하드웨어 설계를 분석하여 사건 구동형 시뮬레이션 (Event-driven simulation) 으로 설계하고 소프트웨어로 구현하였다. 시뮬레이션의 시간적 정확성을 얻기 위하여 상용 낸드 플래시 칩을 선정하여 플래시 메모리 연산들의 시간지연 분포를 구하고 시뮬레이션 동작의 시간지연을 모사하였다. 또한 오류 환경의 유사성을 얻기 위하여 낸드 플래시 메모리 에서 발생 가능한 오류 상황을 모두 분류한 오류 모델을 바탕으로 제어기의 동작과 비동기적으로 발생하는 외부 전원 오류와 동기적으로 발생하는 내부 플래시 오류 발생을 모사하였다. 본 시뮬레이션의 시간적 정확성과 오류 환경의 유사성을 통하여 목표 저장장치의 성능평가와 신뢰성 테스트를 일반 PC 환경에서 수행할 수 있어 플래시 메모리 기반 저장장치 개발의 효율을 높일 수 있다.

제시하는 무순서 플래시 메모리 제어기 시뮬레이션을 검증하기 위하여 FPGA 상에 구현한 시제품과 성능을 비교하였다. 동일한 설정과 작업 부하를 통하여 시제품과 시뮬레이션의 성능을 비교한 결과 99.6%의 동일함을 보였다.

주요어 : 플래시 메모리 기반 저장장치, 플래시 메모리 제어기, 플래시 변환 계층(FTL), 실시간 시뮬레이션

학번 : 2013-20797

목차

그림 목차

표 목차

제 1 장 서론

1.1 연구 동기

전자적 방식으로 동작하는 플래시 메모리는 기계적 방식으로 동작하는 하드디스크에 비하여 고속접근성, 전력효율성, 내구성, 병렬성, 집적성 등의 장점이 있다. 또한 플래시 메모리의 생산기술이 발전함에 따라 용량은 상승하고 가격은 하락하는 추세가 맞물려 플래시 메모리는 저장장치분야에서의 영역을 지속적으로 확대하고 있다. 플래시 메모리의 생산기술은 낸드 칩의 단위 셀의 크기를 작게 만드는 제조공정의 세밀화, 단위 셀에 여러 비트를 저장하는 셀 활용의 세밀화, 수평 및 수직구조로 셀을 적층하는 구조설계의 세밀화를 축으로 발전하여 낸드 플래시 칩의 용량을 크게 늘리면서도 가격은 하락시켰다. 이를 바탕으로 플래시 메모리 기반 저장장치는 전력효율과 소형화가 중요한 모바일 영역부터 높은 성능과 큰 입출력 대역폭을 요구하는 서버영역까지 저장장치시장에서 기존의 하드디스크를 대체하며 응용분야를 넓혀가고 있다.

플래시 메모리 기반 저장장치의 응용분야가 확대되고 수요가 증가함에 따라 저장장치의 개발시 개발환경의 편의성과 저장장치의 신뢰성이 함께 요구되고 있다. 플래시 메모리 기반 저장장치는 플래시 메모리가 가진 고유의 물리적 제약 완화, 동작시 발생하는 오류 처리, 블록장치 인터페이스 제공을 위해 소프트웨어 계층인 플래시 변환 계층(FTL: Flash Translation Layer)을 장치의

내부에서 사용하는데, 이에 따라 플래시 메모리 기반 저장장치의 개발은 저장장치 하드웨어의 설계와 FTL 소프트웨어의 설계가 맞물려 진행된다. 이러한 환경에서는 하드웨어의 성능과 동작특성을 확인하기 위해 목표 저장장치의 시제품을 제작해야 하는데 보드 상에서의 소프트웨어 개발과 신뢰성 검사를 위한 추적은 응용프로그램의 개발에 비해 제한적일 수 밖에 없다. 하드웨어의 설계시 시제품을 제작하고 소프트웨어 개발시 오류를 추적해야 요구사항이 공존하는 상황에서 평가보드를 기반으로 하는 기존의 환경은 개발에 제약이 많고 테스트에 어려움이 많다. 따라서 편의성 측면에서는 제약완화와 오류처리를 담당하는 FTL 소프트웨어의 개발을 하드웨어에 구애받지 않고 독립적으로 진행할 수 있고, 저장장치의 신뢰성 측면에서는 FTL 소프트웨어를 발생가능한 다양한 오류상황에 대해 개발단계에서 충실히 테스트할 수 있는 개발환경이 필요하다.

내장형 소프트웨어 개발에 있어 소프트웨어가 내장될 환경의 시뮬레이션을 구현하여 개발에 활용한다면 소프트웨어 개발의 편의성 제공을 통한 개발에 소요되는 시간적, 물질적 자원 절약에 기여할 수 있고 물리적 제약없는 테스트를 통한 제품의 신뢰성 향상에 기여할 수 있다. 플래시 메모리 기반 저장장치에 내장되는 FTL 역시 내장되어 동작하는 환경과 동일한 환경의 시뮬레이션을 구현하여 개발에 활용한다면 편의성을 향상을 통한 동일한 자원 절약과 목표 저장장치의 신뢰성 향상 효과를 기대할 수 있다.

1.2 연구 내용

본 논문에서는 FTL 의 개발 및 신뢰성 테스트를 위한 무순서 플래시 메모리 제어기와 기반 동작환경을 시뮬레이션으로 구현한다.

무순서 플래시 메모리 제어기는 FTL 로 부터 전달받은 플래시 요청에 대한 처리를 전담하여 FTL 의 부담을 줄이고 플래시 연산 지연시간과 같은 플래시 칩의 특성과 플래시 버스의 개수 및 버스당 칩의 개수와 같은 버스 연결 구조를 고려하여 다수개의 플래시 칩을 최대한 병렬적으로 활용하는 플래시 제어기 이다.

이러한 무순서 플래시 메모리 제어기를 시뮬레이션 하기위해 실제 하드웨어의 모델링, 동작 시간의 모사, 오류 발생의 세 가지 측면에서 모델링하였다. 성능 이슈와 관련하여 플래시 메모리 제어기의 스케줄링 정책, 낸드 플래시 칩의 동작 별 지연시간 정보, 플래시 칩과 플래시 메모리 제어기의 버스 연결구조 등을 모사하였고, 멀티코어 환경에서 성능계수기(Performance counter)를 반복하여 확인하는 쓰레드를 코어에 전담시켜 시간지연 효과의 정확성을 확보하였다. 신뢰성 이슈와 관련해서는 외부 전원오류와 내부 플래시오류를 발생시키는 모듈을 구현하여 모사하였다. 이를 바탕으로 내부 사건의 발생에 따라 다음 동작이 결정되는 사건 구동형(Event-driven) 무순서 플래시 메모리 제어기의 시뮬레이션을 설계하고 구현하였다.

구현한 시뮬레이션을 검증하기 위하여 모사 대상 무순서 플래시 메모리 제어기의 시제품과 동일한 작업부하에 따른 동작시간의 차이를 실제 시간을 기준으로 비교하였다. 동일한 버스 및 칩

설정과 작업부하를 이용하여 무순서 플래시 메모리 제어기의 시제품과 시뮬레이션의 동작시간을 비교해본 결과 99.6%의 유사성을 보였다. 실제 칩과 동일한 동작시간을 갖는 플래시 메모리와 실제 제어기와 동일한 요청순서를 내는 본 시뮬레이션을 통해 개발중인 FTL 을 적용한 가상의 저장장치에 대한 성능 측정과 신뢰성 테스트를 소프트웨어를 통해 수행 할 수 있다.

1.3 논문의 구성

본 논문의 구성은 다음과 같다. 2 장에서는 배경지식으로 플래시 메모리와 저장장치의 구성요소와 동작, 무순서 플래시 메모리 제어기, 그리고 사건 구동형 시뮬레이션(Event-driven simulation)에 대하여 설명한다. 3 장에서는 무순서 플래시 메모리 제어기 시뮬레이션의 설계와 구현에 대하여 설명한다. 4 장에서는 구현한 시뮬레이션과 모사 대상 무순서 플래시 메모리 제어기의 시제품을 동작 시간을 기준으로 비교한 실험을 수행하고 결과를 고찰한다. 마지막으로 5 장에서 결론을 맺는다.

제 2 장 배경 지식 및 관련 연구

2.1 플래시 메모리

플래시 메모리는 전기적으로 내부 데이터를 관리하는 EEPROM 의 한 종류로 전기적 동작을 통해 쓰기, 읽기, 그리고 지우기 동작을 전제조건으로 다시 쓰기가 가능한 비휘발성 메모리이다. 플래시 메모리는 내부 소자의 구성에 따라 NOR 플래시 메모리와 NAND 플래시 메모리로 나누어진다. NOR 플래시 메모리는 기존의 SRAM 과 동일한 인터페이스, 빠른 임의 접근 속도를 제공하여 기존의 내장형 시스템의 코드 보관용으로 사용되어온 ROM 을 대체하고 있다. NAND 플래시 메모리는 NOR 플래시 메모리에 비해 단위 용량 당 가격의 경쟁력과 상대적으로 빠른 쓰기 속도를 제공하여 기존의 내장형 시스템의 데이터 저장용으로 사용되어온 하드디스크를 대체하고 있다. 본 논문에서는 플래시 메모리에 기반한 대용량 데이터 저장장치 시스템에 초점을 맞추고 있으므로 플래시 메모리에 대한 논의를 NAND 플래시 메모리로 한정하여 진행한다.

2.1.1 NAND 플래시 메모리

NAND 플래시 메모리는 스마트폰, 태블릿 PC, USB 드라이브와 같은 소형 내장형 시스템의 데이터 저장매체로 사용되기 시작하여 SSD 드라이브와 같은 중형 개인용 컴퓨터의 대용량 저장장치와

고속 인터페이스를 통해 동작하는 대형 기업용 서버의 데이터 저장매체로 사용영역이 확장되었다.

NAND 플래시 메모리는 데이터 저장 측면에서의 장점을 통하여 사용영역의 확장을 이끌었지만 신뢰성 문제를 일으키는 데이터 관리 측면에서의 단점이 함께 존재한다.

페이지와 페이지들의 집합인 블록 단위로 구성되는 NAND 플래시 메모리는 기존의 메모리와 같이 동일한 주소에 데이터를 덮어쓰기 할 수 없고 지우기 연산이 이뤄 진 후에 쓰기가 가능하다. 쓰기 동작은 페이지 단위, 지우기 동작은 블록단위로 이뤄지기 때문에 플래시 메모리를 이용한 대용량 저장장치는 복잡한 내부 데이터 관리 방법이 필요하다.

제조시 수율을 높게 유지해 가격경쟁력의 우위를 점하기 위해 NAND 플래시 메모리는 생산과정에서의 불량블록을 허용하고있고 이를 초기불량블록이라고 한다. 또한 초기에 정상인 블록도 사용과정에서 일정 횟수의 쓰기와 지우기 연산을 거치면서 불량블록이 될 확률이 높아지는데 저장장치 사용중 발생한 불량블록을 동작중 블량블록이라고 한다. 안정적인 저장장치의 사용을 위해서는 이러한 불량블록들의 관리와 정상블록들에 대한 쓰기-지우기 횟수의 균등화 방법이 필요하다.

NAND 플래시 메모리에 정상적으로 기록된 데이터도 다양한 원인에 의해 비트 반전 오류가 발생할 수 있다. 플래시 메모리 자체는 메모리의 기능만 하기 때문에 비트 반전 오류의 검출과 정정을 위한 하드웨어 또는 소프트웨어의 지원이 필요하다.

NAND 플래시 메모리의 단일 칩 성능만을 활용해서는 고성능-

대용량 저장장치의 요구사항을 만족시키기 어렵다. 칩 하나에서 얻을 수 있는 데이터 전송속도와 내부 연산의 지연 시간은 빠르지 않기 때문이다.

최근의 NAND 플래시 메모리는 공정의 미세화와 생산기술의 발전으로 가격과 용량으로 대표되는 데이터 저장측면의 장점이 두드러지고 있지만, 동시에 신뢰성과 복잡성으로 대표되는 데이터 관리측면의 단점 또한 두드러지고 있고 나아가 사용시의 새로운 제약사항들과 문제점들이 나타나고 있다.

MLC NAND 플래시 메모리는 모든 페이지들을 동일한 성능으로 접근할 수 없고 동일한 신뢰성을 제공하지도 않는다. 짝 페이지 또는 형제 페이지 관계에 있는 페이지들 중 높은 번호의 페이지에 대한 플래시 연산 수행 중 시스템의 전원공급이 중단되는 상황이 발생하면 형제 페이지의 무결성에 문제가 생길 수 있다.

추가적으로 MLC NAND 플래시 메모리의 경우 번호순서대로 페이지를 써야하며 연산 진행과정에서 해당 연산과 무관한 페이지의 데이터가 영향을 받는 문제도 나타난다.

NAND 플래시 메모리의 이러한 특성들에 대해 장점들은 극대화하고 단점들은 최소화 하기 위한 하드웨어-소프트웨어 측면의 접근이 이뤄지고 있다.

2.1.2 NAND 플래시 메모리 내부구조

NAND 플래시 칩은 여러개의 블록으로 구성되고, 블록은 여러개의 페이지로 구성된다. 페이지 내부는 사용자 데이터를

저장하는 데이터 영역과 사용자 데이터와 연관된 메타데이터를 저장하는 예비(Spare)영역으로 구성된다. 데이터 영역의 크기는 일반적으로 디스크의 섹터 크기의 배수이고, 예비 영역의 크기는 일반적으로 데이터영역 내 섹터 당 16 바이트이고 메타데이터의 크기가 증가함에 따라 예비영역의 크기도 증가하는 추세이다.

읽기와 쓰기의 기본 단위인 페이지는 SLC 의 경우 2KB, MLC 의 경우 4KB, 8KB 의 크기를 가진다. 소거의 기본 단위인 블록은 SLC 의 경우 64 개의 페이지, MLC 의 경우 128 개, 256 개가 포함된다. 이러한 구성은 제조사와 소자의 종류에 따라 상이 할 수 있다.

2.1.3 NAND 플래시 메모리 연산

NAND 플래시 메모리의 연산은 페이지 쓰기, 페이지 읽기, 그리고 블록 소거 연산을 기본으로 한다. 이 세 가지 플래시 메모리 연산은 모두 칩 내부의 동작과 앞뒤의 개시단계, 종료단계가 모여 하나의 원자적인 플래시 메모리 연산을 이룬다. 칩 내부의 동작은 쓰기, 읽기, 소거 연산 모두 필요하다. 읽기 연산은 대상 페이지에 해당하는 플래시 내부 트랜지스터의 상태를 읽어 페이지 레지스터로 전달해야 하고, 쓰기 연산과 소거연산의 경우 대상 페이지에 해당하는 플래시 트랜지스터의 상태를 바꿔야 하기 때문이다. 또한 데이터의 흐름 측면에서는 읽기 연산과 쓰기 연산은 동작에 읽거나 쓸 데이터를 해당 칩의 페이지 레지스터에 전달하는 시간이 필요하다. 이러한 구조적인 이유로 NAND 플래시 메모리의

연산은 개시단계, 칩 내부동작, 그리고 종료단계의 순서로 진행된다.

블록 소거 연산은 블록 내의 모든 트랜지스터를 1 의 상태로 만든다. 소거 연산은 소거 명령에서 시작되며 칩 내부동작의 지연시간은 약 2ms 정도이다. 이후 소거 동작이 정상적으로 완료되었는지 확인하는 상태 확인 명령 플래시 칩으로 전달되어 발생가능한 오류정보들을 보고한다.

페이지 쓰기 연산은 소거된 후 준비상태인 블록에 포함된 페이지에 데이터를 쓴다. 개시 단계에서는 대상 플래시 칩의 내부 페이지 레지스터로 플래시 버스를 통해 데이터를 전달한 후 프로그램 명령과 프로그램 대상 페이지의 주소 정보가 플래시 칩으로 함께 보내진다. 프로그램 명령에 의해 데이터는 페이지 레지스터에서 내부 트랜지스터로 전달된다. 프로그램 지연 시간동안 해당 칩은 내부 동작상태(Busy State)로 진입한다. 칩 내부 동작이 끝난 후 종료 단계에서 상태 확인과정이 소거 명령과 동일하게 진행 된다. 쓰기 연산의 지연시간은 SLC 의 경우 200us, MLC 의 경우 800us 정도 이다.

페이지 읽기 연산은 페이지에 적힌 데이터를 읽는다. 개시 단계에서는 읽기 명령과 대상 페이지 주소가 플래시 칩으로 전달된다. 플래시 칩은 내부 동작을 통해 대상 페이지의 데이터를 페이지 레지스터로 전달하고 지연시간이 끝나면 내부 페이지 레지스터에서 데이터가 읽히게 된다. 읽기 연산의 지연시간은 SLC 의 경우 20us, MLC 의 경우 50us 정도 이다.

이 밖에 단일 칩의 성능 향상과 활용성을 높이기 위한 고급 연산들로 페이지 복사 연산, 캐쉬 연산, 그리고 다중 플레인 연산이 있다.

페이지 복사 연산은 한 페이지의 데이터를 외부에서 받은 데이터와 함께 같은 칩 내의 다른 페이지로 복사하는 연산이다. 같은 동작을 기본 동작인 읽기과 쓰기 연산의 조합으로 수행하려고 하면 읽기 연산을 통해 데이터를 읽고, 읽은 데이터를 변경하고, 변경한 데이터를 페이지 쓰기 연산으로 써야한다. 이런 경우 불가피하게 읽기 연산과 쓰기 연산의 과정에서 플래시 버스를 통한 데이터 전송이 2 번 일어나게 된다. 페이지 복사연산은 이러한 중복을 없앤 연산으로 버스를 통해 데이터를 읽고 다시 쓸 필요가 없어 특정 페이지의 내용을 바꿀 경우 더 효율적이다. 페이지 복사 연산은 대상 페이지의 내용을 페이지 읽기 명령에 의해 페이지 레지스터로 전달하고 외부에서 가져온 데이터를 바탕으로 페이지의 일부를 변경한다. 이후 변경된 페이지의 내용은 곧바로 페이지 쓰기 명령을 통해 페이지에 써지고, 상태 확인 명령을 통해 정상적으로 써졌는지 확인한다.

캐쉬 연산은 추가의 페이지 레지스터인 캐쉬레지스터를 도입하여 칩과 페이지 레지스터, 페이지 레지스터와 버스의 동작이 중첩되어 수행될 수 있게 하는 연산이다. 플래시 연산들이 부분적으로 중첩되어 실행될 수 있게 하므로써 플래시 연산의 지연시간의

일부가 감추어질 수 있다. 캐쉬 연산의 이점을 누리기 위해선 연속적인 캐쉬 연산의 블록 주소가 동일해야 하고, 블록 내 페이지 번호가 1 씩 증가해야 하는 제약에 따라야 한다. 하지만 대량의 연속적인 플래시 연산이 수행되는 경우에는 캐쉬 연산을 통하여 효율향상을 극대화 시킬 수 있다.

다중 플레인 연산은 하나의 플래시 메모리 다이를 이루는 여러개의 플레인과 페이지 레지스터 사이의 데이터 전달을 병렬적으로 수행하여 성능을 향상 시키는 연산이다. 이 연산은 한번의 플래시 연산 지연 시간에 의해 여러 플레인의 페이지 레지스터와 셀 배열 사이에서 데이터를 동시에 전달할 수 있다. 따라서 플래시 연산 지연시간이 긴 경우 다중 플레인 연산을 통해 성능향상을 효과를 얻을 수 있다. 각 플레인에 별도로 존재하는 페이지 레지스터와 공유하는 외부 인터페이스에 의해 데이터처리는 병렬적으로 이뤄지고 전송은 직렬화되어 제공된다.

2.2 플래시 메모리 기반 저장장치

전형적인 플래시 메모리 기반 저장장치를 이루는 주요 구성요소는 FTL, 플래시 메모리 제어기, 그리고 플래시 메모리 칩이다.

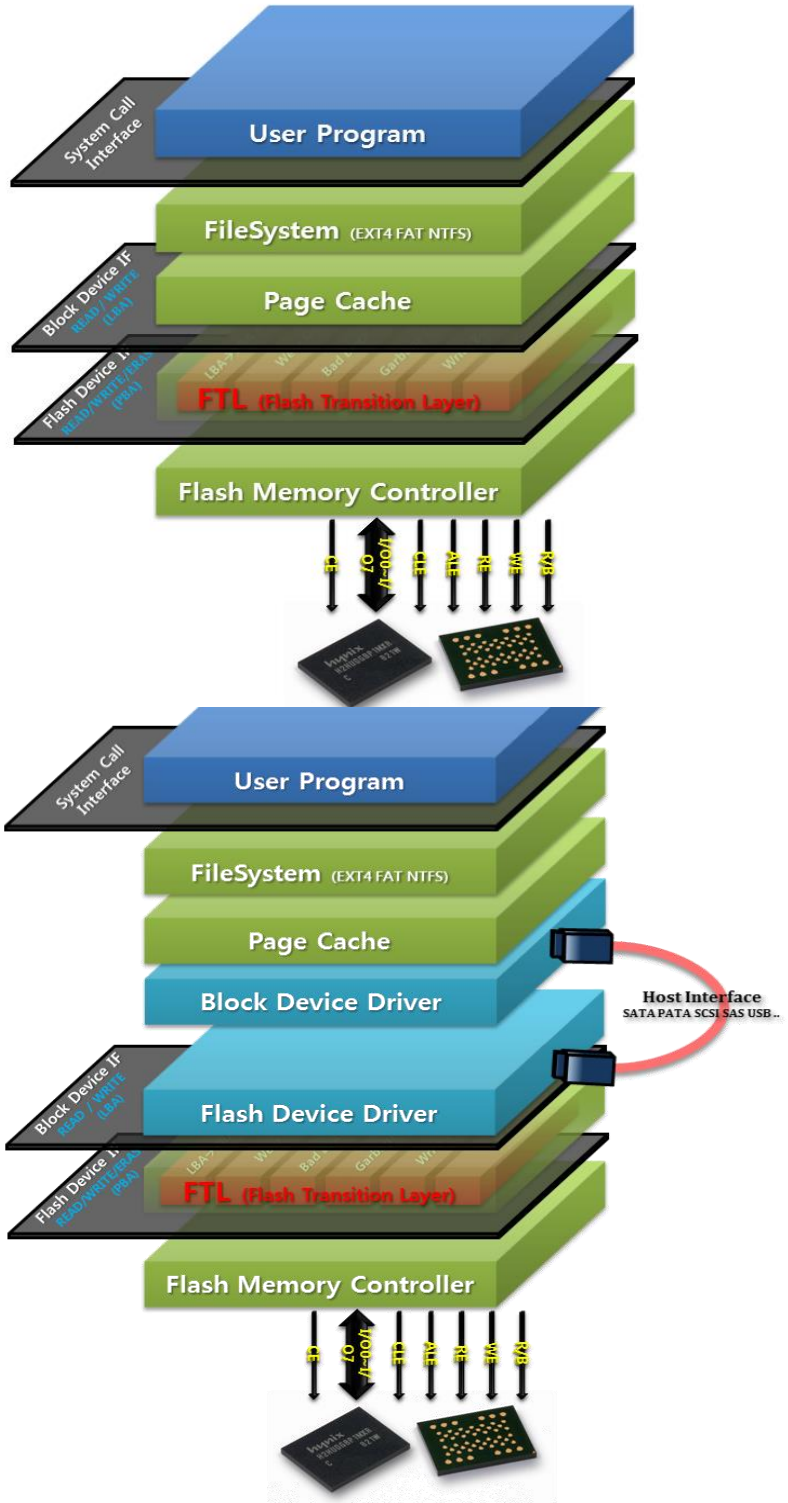

그림 1. 플래시 메모리 기반 내장형 블록장치와 독립형 블록 장치

FTL 은 호스트 시스템이 저장장치를 바라보는 관점인 블록 장치 인터페이스와 실제 플래시 메모리가 제공하는 관점인 플래시 메모리 인터페이스를 일치시키기 위한 사상정보를 관리한다. 호스트시스템에서 블록장치의 논리적 블록 주소에 대해 요청한 읽기 및 쓰기 요청을 처리하기 위하여 FTL 은 플래시 칩의 물리적 페이지 주소에 대한 일련의 플래시 연산들로 변환한다.

플래시 메모리 제어기는 하드웨어로 구현되어 플래시 메모리 칩과 연결되고 데이터의 저장 및 인출과 관련된 칩들의 동작을 관리한다.

실제 데이터를 저장하는 플래시 메모리 칩은 규정된 인터페이스에 의해 플래시 메모리 제어기와 연결된다.

플래시 메모리 저장장치는 사용환경에 따라 세 가지 구성요소를 다양한 위상으로 배치시켜 구현할 수 있다.

내장형 시스템에서 많이 사용되는 그림 1 상단의 구성은 FTL 을 비롯한 모든 저장장치의 구성 요소들이 호스트시스템의 프로세스의 제어를 통해 동작한다. 이러한 구성에서 FTL 은 상위 시스템 소프트웨어인 파일시스템, 가상메모리 시스템, 그리고 데이터베이스 관리시스템 등에게 블록장치인터페이스를 제공한다. 플래시 메모리 제어기는 호스트의 프로세서와 연결되는데 메모리접근 인터페이스를 통하거나, 내장형 프로세서 내부에 플래시 메모리 제어기가 함께 들어가는 경우도 있다.

그림 1 하단과 같이 일반 컴퓨터에서 많이 사용되는 독립형 플래시 저장장치인 SSD 의 구성은 제어기와 FTL 이 모두 저장장치 내에 구현되고 저장장치와 호스트시스템은 규격화된 블록장치 인터페이스를 통해 연결된다. 이러한 구성은 기존의 호스트시스템에 저장장치로 통합하기 용이하고 호스트시스템은 전통적인 인터페이스로 동일하게 플래시 기반 저장장치를 사용할 수 있다.

2.3 저장장치 시뮬레이션 관련연구

2.3.1 이산사건 시뮬레이션

모의실험 등으로 불리는 시뮬레이션은 실존하는 시스템을 직접활용하는 대신 대상 시스템을 수학적 모형을 사용하여 여러가지 가상실험을 수행하는 것이다. 따라서 시뮬레이션은 크게 대상 시스템의 모델링과 모델을 이용한 실험으로 나누어 진다.

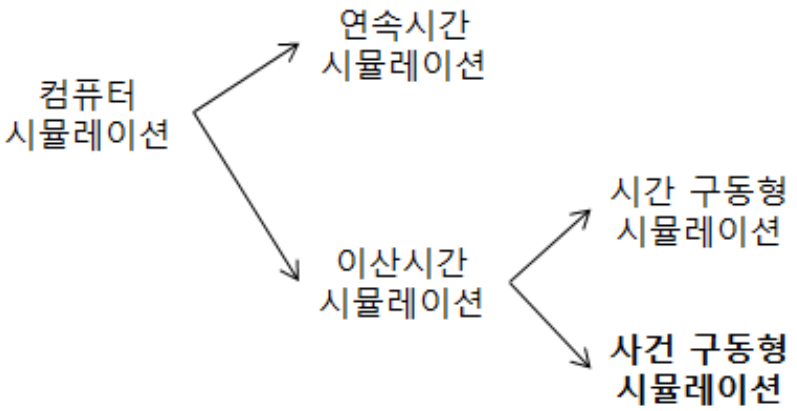

그림 2. 컴퓨터 시뮬레이션

시뮬레이션은 실제 시스템을 이용하는 대신 컴퓨터 모델을 사용하므로써 실제 시스템을 구성하는데 드는 인력과 경비를 절약하고 시스템에 영향을 미치는 다양한 변수들을 자유롭게 설정할 수 있다는 장점이 있다. 따라서 시뮬레이션 대상 시스템의 자세한 모델을 활용하여 결과를 평가므로써 신뢰성 있는 결과를 도출 할 수 있다.

시뮬레이션은 구동하는 그림 2 와 같이 진행기준에 따라 시간에 따른 구동과 사건에 따른 구동방식이 있다.

시간에 따른 구동방식은 수학적으로 모델링된 미분방적식 모델을 수치해석하는 연속시간 시뮬레이션과 상태 다이어그램으로 모델링된 유한상태기계 모델을 해석하는 이산시간 시뮬레이션으로 나눌 수 있다. 시간에 따른 구동방식은 시뮬레이션을 일정한 시간 간격으로 실행시키면서 모델의 수치와 상태를 해석한다.

사건에 따른 구동방식은 이산사건 시뮬레이션으로 불리며 시뮬레이션의 대상 시스템의 외부 혹은 내부에서 유의미한 '사건'이 일어났을 때에만 모델을 실행시킨다. 사건은 실제 시스템의 동작을 시뮬레이션의 목적에 따라 추상화 한 신호로, 이산사건이란 임의의 시각에 불규칙적으로 일어나는 사건을 의미한다. 이산사건 시스템의 모델링은 시스템을 여러개의 개념적인 부모델을 실제 시스템에 대응되는 구성요소들로 명세하고 내외부에서 발생하는 사건들을 식별하는 방식으로 설계된다.

실제 저장장치 시스템은 외부에서 유입되는 요청들과 요청들을 처리하는 과정에서 내부 요소들 사이에 발생하는 다양한 사건들의 추상화만으로도 충분한 수준의 시뮬레이션을 구현할 수 있다. 저장장치 시뮬레이션을 구현하는데 있어 내부 요소들의 개념적 모델과 사건들을 식별하므로써 최소한의 모델링 노력과 성능부하로 가상실험의 높은 신뢰성을 제공하는 저장장치 시뮬레이션을 구현할 수 있다.

2.3.2 디스크 저장장치 시뮬레이션 관련연구

저장장치의 성능평가 혹은 저장장치를 이용하는 소프트웨어의 성능평가를 위하여 저장장치의 타이밍 모델에 기반한 시뮬레이션이 사용된다. 저장장치 타이밍모델은 실제 저장장치인 블록 장치와

동일한 인터페이스를 제공하면서 블록장치 요청에 대한 처리와 함께 요청의 처리 지연시간을 반환한다. 저장장치 시뮬레이션은 이러한 타이밍 모델을 바탕으로 저장장치의 처리 지연시간을 모사하여 실제 저장장치를 이용하는 것과 같은 환경을 제공한다.

디스크 시뮬레이션에 관한 연구[1]는 장치드라이버, 버스, 제어기, 어댑터, 디스크를 가정한 보조저장장치 내부 구성요소들을 소프트웨어 모듈로 구현하였고 대규모 시스템 수준의 시뮬레이터와의 연동을 위한 고리를 제공하는 사건 구동형 디스크 저장장치 시뮬레이션이다. 그러나 시뮬레이션의 결과를 가상시간으로 제공하기 때문에 실제 시스템에 온라인으로 연결되어 요청을 처리하는 방식으로는 사용할 수 없어 주로 다른 시뮬레이션에서 타이밍 모델을 이용하기 위한 목적으로 연동시켜 사용한다.

실시간 온라인 디스크 저장장치 시뮬레이션에 관한 연구[2]는 디스크 저장장치와 호스트 시스템을 별도의 컴퓨터로 분리하고 블록장치 인터페이스로 연결하여 호스트 시스템에서 실제 블록장치를 이용하는 것과 같은 시뮬레이션 환경을 제공하였다. [1]의 타이밍 모델을 사용하였고 실제 동작 성능을 보장하기 위하여 저장장치로 구현된 컴퓨터에서 동작하는 저장장치 시뮬레이터가 DRAM 상에서 캐싱되는 크기만큼의 저장장치 용량을 제공하였다. 저장장치 시뮬레이션과 호스트시스템을 별도의 기계로

분리시켜 호스트 시스템의 동작에 의한 시뮬레이션의 불확실성을 제거하려고 시도하였지만 저장장치 시뮬레이션이 동작하는 컴퓨터에서 시스템 소프트웨어의 간섭으로 인한 불확실성에 대한 언급은 없었다.

2.3.3 플래시 메모리 시뮬레이션 관련연구

플래시 메모리와 그에 기반한 저장장치의 성능평가 및 신뢰성 평가를 위하여 플래시 메모리의 시뮬레이션을 사용할 수 있다. 성능평가를 위한 시뮬레이션은 기존의 디스크 시뮬레이션에 사용되었던 타이밍모델에 플래시 메모리 모델을 적용하여 사용하거나 플래시 메모리 칩 수준의 물리적 요소들을 세밀히 모델링하여 새로운 플래시 메모리 시뮬레이션이 구현되기도 하였다.

[3]은 기존의 디스크 시뮬레이션에 사용된 환경에서 타이밍모델을 플래시 메모리에 맞게 수선하여 구현되었다. 기존의 디스크 시뮬레이션과 마찬가지로 실시간 저장장치 성능비교를 위한 용도 보다는 모델에 기반한 요청의 처리시간을 구할 수 있다.

[4]은 플래시 메모리 저장장치의 칩-버스 수준 타이밍 모델과 기본 플래시 연산들과 고급 연산들을 동작상태 다이어그램으로 모델링 하여 내부 모델의 시간지연을 반영한 시뮬레이션을 구현하였다. 내부 칩과 버스의 위상과 실제 플래시 메모리 칩으로

부터 측정한 타이밍 모델을 바탕으로 연산의 진행에 따른 세부적인 상태전이를 모델링하여 정확한 플래시 메모리 저장장치의 요청 처리결과를 반환하였다.

[5]은 호스트시스템을 가상머신으로 가정하고 시뮬레이션이 구동되는 실제 호스트시스템의 램 디스크를 이용하여 플래시 메모리 기반 저장장치의 사용환경을 가상머신에 제공하였다. 호스트시스템 내에서 소프트웨어적으로 수행되는 가상머신을 이용하여 가상의 저장장치 동작의 상태를 실시간으로 파악하고 성능을 평가하는데 이용하였다.

디스크 및 플래시 저장장치의 시뮬레이션에 관련된 연구는 크게 정확한 타이밍 모델을 제공하는 것과 대상 저장장치의 성능 및 신뢰성 특성을 반영하여 실제 호스트시스템에 블록장치 디바이스로 제공하는 방향으로 진행되어왔다. 하지만 성능평가의 목적인 경우 저장장치 시뮬레이션 동작의 시간적 정확성 제공이 제한적인 경우가 많고 신뢰성평가 목적으로는 연구되어있는 저장장치의 시뮬레이션이 없는 실정이다.

본 논문에서 제안하는 무순서 플래시 메모리 제어기의 시뮬레이션을 통하여 성능과 신뢰성을 평가할 수 있는 환경을 제공할 수 있을 것이다.

제 3 장 무순서 플래시 메모리 제어기

3.1 플래시 메모리 연산 실행 모델

하나의 디스크 헤드를 통해 데이터를 읽고 쓸 수 있어 병렬성이 제한되는 하드디스크와 달리 플래시 메모리를 기반으로 하는 저장장치는 다수의 칩들을 독립적으로 동작시킬 수 있기 때문에 활용할 수 있는 병렬성이 높다. 연산의 수행 제약에 따라 다수의 칩을 병렬적으로 활용하는 방법은 다양하게 나뉠 수 있다.

플래시 칩을 동작시키는 관점에서 전체 칩의 개수는 제어기에서 칩으로 데이터를 전송하는 버스의 수와 버스를 공유하는 칩들의 개수로 결정된다. 데이터를 처리하는 플래시 메모리 칩들은 독립적으로 동작할 수 있지만 데이터를 전송하는 버스는 칩들간에 공유되기 때문에 플래시 메모리 칩을 병렬적으로 활용하는 문제는 데이터를 전송하는 버스를 활용하는 문제와 버스를 공유하는 다수의 칩을 활용하는 문제로 분리된다.

데이터를 전송하는 플래시 메모리 버스는 독립된 칩들과 연결되어 있다. 따라서 플래시 메모리 버스에 연결된 칩들을 활용하는 방법은 한 번에 하나의 연산으로 처리 하므로써 플래시

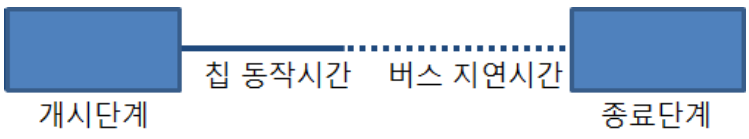

그림 3. 단일 플래시 연산

버스를 순차적으로 동작시키는 방법과 여러 플래시 연산에 대해 여러 버스를 함께 동작시키는 병렬적 실행방법으로 분리된다. 플래시 버스를 순차적으로 동작시키는 버스에 연결된 플래시 칩들 중 한번에 하나의 칩만 동작하기 때문에 칩을 병렬적으로 사용하는 경우에 대해서는 고려할 필요가 없다. 한편, 버스를 공유하는 플래시 메모리 칩들을 활용하는 방법은 동작의 중첩 여부와, 개시와 완료순서의 일치 여부에 따라 순차실행, 순서중첩실행, 그리고 무순서 중첩 실행 모델로 정의된다. 플래시 메모리 연산의 실행모델은 프로세서에서 명령어를 실행하는 모델과 유사하다.

무순서 플래시 메모리 제어기의 설계시 다수개의 플래시 메모리 연산을 수행하는 경우 칩의 지연시간 동안 공유되는 버스를 활용하기 위하여 전체 플래시 연산에서 개시단계와 종료단계를 분리하였다. 두 단계를 구분하게 될 경우 칩만 활용하는 내부 동작과 칩과 버스를 활용하는 개시 및 종료 단계를 구분하여 버스와 칩 활용의 병렬성을 높일 수 있다.

플래시 연산을 나누어 처리하는 경우 나누어진 연산의 처리 순서에 관한 문제가 새롭게 등장한다. 제어기의 처리 방식에 따라 구분된 연산들의 처리를 원래 연산의 순서와 동일하게 할 수도 있고 반대로, 순서를 강제하지 않을 경우 칩의 처리 시간과 버스의 활용상태에 따라 처리 순서를 원래 연산의 순서에 상관없이 할 수도 있다.

무순서 플래시 메모리 제어기 설계에서는 플래시 연산을 개시와 종료단계로 나누어 버스를 공유하는 칩들 사이에 다수의 연산을 중첩 실행하는 기준과, 요청된 플래시 연산의 순서와 단계별 개별 연산의 순서를 일치시켜 실행하는 기준에 따라 순차 실행과, 순서 중첩 실행, 그리고 무순서 중첩 실행 모델로 나누었다[6].

3.2 제어기의 설계 및 구현 목표

무순서 플래시 메모리 제어기는 아래와 같은 설계 및 구현 목표를 설정하였다.

1) 다수개의 플래시 메모리 칩을 최대한 활용하는 병렬성 확보
2) 제어기와 플래시 메모리 관리 소프트웨어 사이의 인터페이스를 정의하여 상호간 독립성 확보
3) 저장장치의 다양한 내부구조 변화에 대응할 수 있는 제어기 모듈성과 확장성 확보
4) 상위의 FTL 과 저장장치 사용환경에 따른 요청 처리 방법의 다양성 확보
5) 실행 제약조건에 따른 실행 모델을 지원하기 위한 제어기 내부구조 모듈성과 확장성 확보

이러한 목표를 달성하기 위하여 무순서 플래시 메모리 제어기의 구현 시 아래와 같은 결정을 하였다.

1) 병렬성 확보를 위하여 플래시 요청들의 처리 순서에 대한 제약을 최소화함으로써 플래시 연산이 효율적으로 처리 될 수 있도록 하였다.
2) 플래시 연산의 효율적인 스케줄링을 위하여 필요한 요청의 종류, 버스와 칩의 상태 등이 가장 잘 관찰되는 하드웨어 단계에서 스케줄링을 전담하게 하였다.
3) FTL 과 플래시 메모리 제어기 사이의 인터페이스를 패킷 인터페이스로 정의하여 상호간의 모듈성, 내부적 확장성, 기능적 재사용성을 높였다. 패킷 인터페이스를 따르는 어떠한 FTL 도 무순서 플래시 메모리 제어기를 이용할 수 있다.
4) 설계한 제어기를 실제 FPGA 상에 구현하므로써 동작의 정확성과 성능의 향상을 확인하였다.
5) 플래시 요청들을 작업단위에 따라 분류한 '스트림' 을 정의하고 스트림간의 우선순위 처리를 지원하여 FTL 이 다수개의 플래시 메모리 칩의 병렬성을 최대한 활용할 수 있게 하였다.

위에서 정의한 무순서 플래시 메모리 제어기 설계와 구현 시의 목표와 결정에 따라 시뮬레이션에서의 설계와 구현 시 반영하였다.

3.3 무순서 플래시 메모리 제어기의 구조

무순서 플래시 메모리 제어기는 하드웨어로 구현되어 플래시 버스와 칩의 상태를 확인하며 요청들을 스케줄링 하기 때문에 버스와 칩을 효율적으로 사용할 수 있다. 요청의 스케줄링을 소프트웨어가 담당하는 하드디스크와 달리 플래시 메모리 제어기를 하드웨어로 구현하는 이유는 플래시 메모리 연산이 마이크로초 단위이기 때문에 밀리초 단위인 하드디스크에 비해 가용한 스케줄링 시간이 짧기 때문이다.

FTL 과 무순서 플래시 메모리 제어기는 쌍으로 구성된 입력-출력 선입선출 큐로 연결된다. 각 선입선출 큐를 통과하는 정보는 FTL 이 제어기에 전달하는 플래시 요청 패킷과 제어기가 FTL 에 전달하는 플래시 응답 패킷이다. FTL 이 제어기에 보내는 모든 연산은 패킷의 형태로 전달된다.

패킷 인터페이스로 분리된 FTL 과 플래시 메모리 제어기는 최대한의 요청들을 추출하는 작업과 최대한의 요청들을 처리하는 작업에 집중할 수 있다. 이를 통해 FTL 은 요청들의 스케줄링은 신경쓰지 않고 사상, 쓰레기 수집, 마모 균등화와 같은 플래시 메모리 관리에 전념할 수 있다.

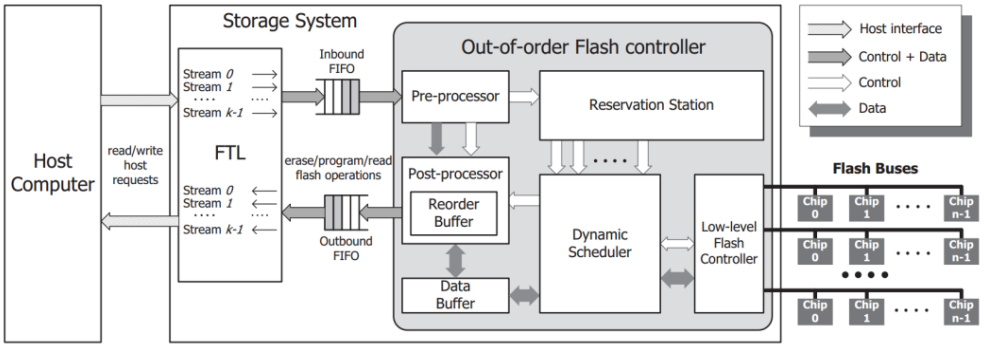

그림 4. 무순서 플래시 메모리 제어기 아키텍처

입력받은 패킷을 해석하여 예약 스테이션으로 이동시킨다. 예약 스테이션에 위치한 요청 패킷은 동적 스케줄러에 의해 선택되어 연산이 수행될 대상 플래시 칩으로 전송될 때까지 대기한다. 쓰기 요청의 경우 헤더와 데이터로 분리되어 헤더는 예약 스테이션으로 데이터는 버퍼로 이동하여 실제 데이터 전송 필요시 버퍼에서 칩으로 전송된다.

동적 스케줄러는 버스와 칩들의 상태를 확인하면서 예약 스테이션에 위치한 요청들을 선택하여 요청을 처리할 대상 칩으로 보낸다. 칩이 연산을 수행하지 않는 준비상태가 되면 동적 스케줄러는 저수준 플래시 메모리 제어기로 개시단계 요청을 전송한다. 칩에서 요청을 처리하여 지연시간이 흐른 후 해당 칩이 다시 준비 상태가 되면 동적 스케줄러는 해당 칩에 종료단계 요청을 보낸다. 개시단계 요청, 플래시 칩 처리, 종료단계 요청이 하나의 전체적인 플래시 요청을 이룬다. 플래시 칩은 처리지연시간 동안에는 다른 칩과 공유하는 버스를 사용하지 않는다. 동적

스케줄러는 이러한 버스와 칩의 상태를 확인하고 요청을 처리하고 있지 않은 다른 칩에 다른 플래시 요청을 보낸다.

플래시 요청들이 처리된 후 위치하는 재정렬 버퍼는 동일 한 스트림에 속한 플래시 요청 패킷이 제어기에 전달된 순서로 FTL 에 응답되도록 재정렬하는 역할을 한다. 이는 동적 스케줄링 프로세서에서의 재정렬 버퍼와 마찬가지로 완료된 연산을 저장하고 응답순서를 요청순서와 동일하게 만들어준다.

제 4 장 무순서 플래시 메모리 제어기 시뮬레이션

4.1 시뮬레이션 설계 및 구현상의 목표

플래시 메모리 제어기 시뮬레이션의 설계 및 구현 대상을 무순서 플래시 메모리 제어기[6]를 모델로 선정하였다. 개발 보드상에 FPGA 로 구현한 무순서 플래시 메모리 제어기는 시뮬레이션의 구현과 동작 결과를 비교할 수 있다. 플래시 메모리 제어기 시뮬레이션의 설계 및 구현 목표로 삼은 점들은 다음과 같다.

(1) 플래시 메모리 제어기의 시뮬레이션은 시간 측면과, 기능 측면으로 나누어 구현되어야 한다. 시뮬레이션을 통해 제어를 모사하게 되는 플래시 메모리는 다양한 성능을 가질 수 있다. 따라서 제어기의 성능을 결정하는 시간 측면과, 결과를 결정하는 기능 측면은 분리되어 구현하여야 한다.

(2) 플래시 메모리 제어기의 시뮬레이션은 실시간으로 동작해야 한다. 실시간으로 동작하는 시뮬레이션을 통해 벤치마크를 통한 저장장치 성능평가에 이용할 수 있다. 이를 위해 시뮬레이션은 실제 플래시 메모리의 칩의 연산 지연시간을 바탕으로 FTL 의 플래시 요청들에 실시간으로 응답해야 한다.

(3) 플래시 메모리 제어기의 시뮬레이션은 플래시 메모리 칩과 전원 차단과 같은 오류동작들을 모사해야 한다. 오류를

모사하는 제어기 시뮬레이션을 이용하여 플래시 메모리 소프트웨어의 신뢰성을 테스트 할 수 있다. 따라서 플래시 메모리의 비동기적 외부오류와 동기적 내부오류를 모사하고 이를 플래시 메모리 소프트웨어에 전달 하므로써 신뢰성을 테스트할 수 있도록 해야한다.

위와 같은 설계 및 구현 목표를 달성하기 위하여 아래와 같은 설계상의 결정을 하였다.

(1) 실시간 동작을 위하여 다중 프로세서 코어 동작환경에서 두 개의 코어를 성능카운터를 확인작업과 데이터전송작업으로 각각 분할하여 활용하였다. 또한 시간을 정밀하게 측정하는데 있어 방해가 될 요소들을 사전에 식별하여 실시간 동작에 영향을 받지 않도록 하였다.

(2) 기존의 하드웨어로 구현된 무순서 플래시 메모리 제어기의 아키텍쳐를 바탕으로 시간과 기능 측면을 분리하여 시뮬레이션 설계에 반영하였다.

(3) 플래시 메모리의 오류상황을 체계적으로 분류한 오류 모델[7]을 바탕으로 오류발생모듈을 작성하였다.

4.2 시뮬레이션 전체 아키텍처

본 절에서는 3 장에서 살펴본 무순서 플래시 메모리 제어기의 실제 아키텍처와 시뮬레이션으로 구현된 아키텍처를 비교하여 4.1 절의 설계 및 구현 상의 목표가 어떻게 반영되는지 살펴보도록 한다.

그림 5 는 플래시 메모리 제어기 시뮬레이터를 저장장치 내부의 관점에서 조망한다. 시뮬레이터를 구성하는 모듈들은 세가지 목적에 의하여 분류된다. 하드웨어 모델링 모듈은 무순서 플래시 메모리 제어기의 실제 하드웨어 동작을 모사하기 위한 모듈들의 집합이다. 시점 시뮬레이션 모듈은 무순서 플래시 메모리 제어기 동작의 시간지연 특성을 모사하기 위한 모듈들의 집합이다. 마지막으로

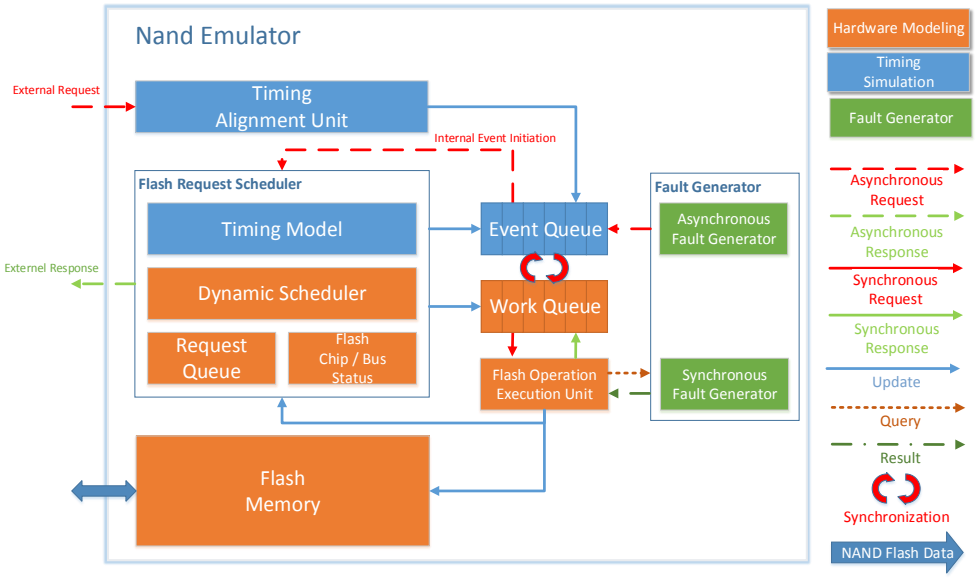

그림 5. 플래시 메모리 제어기 시뮬레이터 아키텍처

오류 발생기 모듈은 플래시 메모리 동작을 기준으로 내외부에서 발생하는 오류들을 모사하기 위한 모듈들의 집합이다. 각 모듈들은 요청과 응답, 질의와 결과반환을 주고 받고 서로간의 정보업데이트와 동기화를 반복하며 시뮬레이션을 진행한다.

4.2.1 시점 시뮬레이션

시점 시뮬레이션 모듈집합은 무순서 플래시 메모리 제어기 동작의 시간지연 특성을 실제시간 지연방식으로 모사한다.

시점 정렬 유닛은 외부에서 들어온 요청에 대해 발생시점을 판단하여 처리요청 시간을 조정한다. 실제 요구되는 처리 시간과 제어기로 입력된 요청의 처리시간의 차이를 보상하기 위한 모듈이다. 시뮬레이션 과정에서 발생 가능한 시간 차이를 보상하므로써 정확도를 높이기 위한 목적이 있다.

사건 큐는 시점 정렬 유닛을 통해 처리 시간이 조정된 요청들이 저장되는 큐이다. 큐에 저장된 요청들은 미리 측정된 플래시 메모리의 동작시간지연에 의해 하나의 요청이 모사하는 동작시간이 결정된다. 특정시점에 외부에서 발생하는 비동기적 오류를 모사하는 경우, 비동기 오류 발생기로부터 오류 발생 요청을 받아 처리한다. 오류 발생 요청을 받으면 사건 큐는 오류 요청을 기준으로 큐

내부의 요청들을 오류 이전의 요청과 오류 이후의 요청을 나누게 된다.

시점모델은 동작을 모사하는 대상 플래시 메모리의 동작시간지연을 미리 측정한 값의 집합이다. 읽기, 쓰기, 소거 등 동작과 칩과 버스를 사용하는 상태의 변화에 따른 단계별 지연시간을 반영한다. 사건 큐의 동작들은 외부에서 유입되어 동적스케줄러에 의해 스케줄링 될때까지 동작 지연시간은 결정되어 있지 않다. 이때 사건큐의 요청들은 시뮬레이션 대상 플래시 메모리의 시점 모델을 참조하여 요청의 시간지연을 결정한다.

4.2.2 하드웨어 모델링

하드웨어 모델링 모듈집합은 무순서 플래시 메모리 제어기의 실제 기능적 동작을 모사한다.

플래시 메모리 모듈은 메모리 상에 모사된 플래시 메모리의 기능적 모델이다. 플래시 메모리로 전달되는 데이터 요청들은 플래시 메모리 모듈이 동작하는 메모리 공간상에서 모사되어 실제 데이터의 저장과 인출의 모사는 메모리의 입출력으로 처리한다.

작업큐는 실제 무순서 플래시 메모리 제어기에서 외부 요청들을 임시로 저장하고 있는 예약스테이션을 모사한다. 작업큐의 요청들은 사건큐와 동기화 되어 요청들의 풀이 유지되고 동적 스케줄러로

부터 정보를 받아 들여 수행가능한 요청을 플래시 동작수행유닛으로 보낸다.

플래시 요청스케줄러는 동적스케줄러, 각 칩별로 유지되는 요청큐, 플래시 칩과 버스의 상태정보로 구성된다. 동적스케줄러는 작업큐에 저장된 요청들 중 수행가능한 요청을 선택하여 해당 칩의 요청큐로 보내어 처리한다. 요청큐는 플래시 동작을 무순서로 수행할 때 칩별로 유지하는 선입선출큐이다. 해당 동작이 완료되었음을 동작수행유닛으로부터 보고받고 외부로 응답을 전송한다.

플래시 동작수행유닛은 작업큐에 있는 요청 중 요청스케줄러에 의해 선택된 요청을 수행하는 동작을 모사한다. 요청 스케줄러에는 버스와 칩의 상태를 갱신하기위한 정보를 전송하고 플래시 메모리 모듈의 정보를 갱신한다.

4.2.3 오류 발생기

오류 발생기 모듈집합은 플래시 메모리의 동작중 발생하는 내외부 오류들을 모사한다.

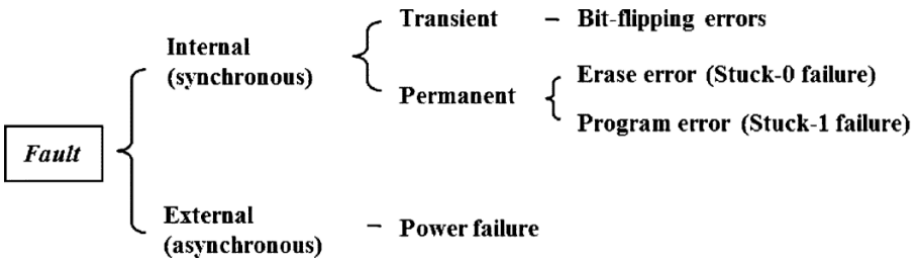

그림 6. Abstract Fault Model

분류한 Abstract Fault Model 을 제시하였다. 본 시뮬레이션의 오류 발생기 모듈은 위의 오류상황 분류를 적용하여 내부적으로 발생하는 동기적 오류와 외부적으로 발생하는 비동기적 오류를 구분하여 모듈집합을 작성하였다.

내부적으로 발생하는 동기적 오류는 오류 발생기의 동기적 오류 발생기 모듈에 의해 발생된다. 동기적 오류 발생기는 플래시 동작 실행유닛이 동작을 실행할때 해당 동작의 오류 여부를 묻는 질의에 대해 오류 확률에 의해 질의 결과를 되돌려준다.

외부적으로 발생하는 비동기적 오류는 오류 발생기의 비동기적 오류 발생기 모듈에 의해 발생된다. 비동기적 오류 발생기는 오류 발생요청을 사건큐로 전송한다. 비동기적 오류 발생 요청을 전달받은 사건큐는 오류 요청을 기준으로 이전 요청과 이후 요청을 결정할 수 있다.

4.3 시뮬레이션 구현

본 시뮬레이션은 플래시 메모리 제어기 동작에 있어서 칩 수준의 모든 동작을 '사건' 으로 다루고 상호 이산적 사건의 발생, 발생으로 인한 시간의 지연, 사건의 발생으로 인한 후속 사건의 인과적 발생으로 모델링 한 이산-사건구동형 고해상 실시간 지연 시뮬레이션으로 설계되었다. 본 절에서는 4.2 절에서 설명한 위의 시뮬레이션 아키텍처의 실제 구현내용을 설명한다.

4.3.1 시점 시뮬레이션

시뮬레이션은 플래시 메모리 제어기의 실제 동작 지연시간을 실제 시간 흐름으로 재연하는 역할을 한다. 이를 위해 실제 동작을 시뮬레이션 내부에서 발생하는 '사건' 으로 정의하고, 지연시간의 수치를 파악하고, 파악한 수치를 시뮬레이션 내부에서 다시 실제 시간의 흐름으로 재연하고, 인과적으로 발생하는 후속사건으로 처리하는 과정을 반복해야 한다.

시점 시뮬레이션은 동작별 지연시간의 분포를 시뮬레이션의 시간지연의 근거자료로 활용하였고 실제 시간 시간흐름의 재연은 프로세서에 내장된 성능카운터를 시뮬레이션의 벽시계로 활용하여 구현하였다.

■ 시뮬레이션 이산사건의 분류

본 시뮬레이션의 시간 흐름과 관계있는 사건은 칩의 내부 동작과 데이터 전송단계로 구분된다. 데이터 전송은 읽은 데이터나 쓸 데이터를 외부 혹은 칩으로 전송하는 단계로 버스와 칩을 모두 사용한다. 칩의 내부 동작은 데이터를 버퍼로 읽어내거나 버퍼의 데이터를 쓰는 단계로 동작을 수행하는 해당 칩만 사용한다.

플래시 칩에서 수행하는 명령은 모두 위의 두 동작으로 구분할 수 있다. 읽기 동작은 페이지에서 버퍼로 데이터를 읽어오는 칩 내부 동작단계와 버퍼로 읽어온 데이터를 버스를 통해 외부로 전송하는 데이터 전송단계로 나눠진다. 쓰기 동작은 페이지에 쓸 데이터를 버스를 통해 버퍼로 전송하는 데이터 전송단계와 버퍼에 전송된 데이터를 페이지에 쓰는 칩 내부 동작단계로 나눠진다. 소거 동작은 데이터 전송 없이 소거 대상 블럭의 값을 모두 1 로 바꾸는 칩 내부 동작으로만 구성된다.

별개의 사건으로 분리된 칩 내부 동작과 데이터 전송 동작은 동적스케줄러에 의해 역시 별개로 스케줄링된다. 동적스케줄러는 버스와 칩의 상태를 보면서 현재 상황에서 수행가능한 동작을 선택하게 된다. 수행 동작의 종료와 함께 전체 시뮬레이션의 상태가 변화하여 새로운 사건이 인과적으로 발생하게 된다.

■ 이산사건의 지연시간 파악

이산사건은 시작시점과 종료시점이 이산적으로 정의된 사건이다. 사건의 지연시간은 이산적으로 정의 된 시작시점과 종료시점의 차이로 정의한다. 실시간으로 동작하는 시뮬레이터는 모사중인 실제 사건에 대하여 실제 시간으로 지연시간이 흐른 후 완료 처리를 해야한다.

앞서 정의한 칩 수준 이산사건의 지연시간은 플래시 칩의 동작 지연시간이다. 플래시 메모리 제어기의 실제 제어 대상 플래시 칩의 동작 지연시간을 측정하여 페이지 읽기 및 쓰기와 블럭 소거 동작의 동작별 지연시간 분포를 구하여 사건 지연시간의 자료로 사용하였다.

실제 플래시 칩의 동작 지연시간을 측정하기 위하여 오픈 플랫폼의 개발보드 상에 장착된 상용 낸드 플래시 칩을 사용하였다.

■ 시간지연의 재연

실제 시간 흐름의 재연은 시간 흐름을 파악할 수 있는 시간의 ‘기준’ 이 있어야 한다. 이산 사건의 시작 시점을 ‘기준’ 으로 정해진 지연시간이 흐른 후 목표 종료시점에 해당 사건의 종료처리를 해야하기 때문이다.

실제 시간의 흐름을 파악하기 위한 기준 시계로 현재 개인용 컴퓨터에서 활용할 수 있는 프로세서의 타임스탬프카운터를 활용하였다. 인텔 프로세서와 칩셋을 기준으로 시간에 따라 일정한

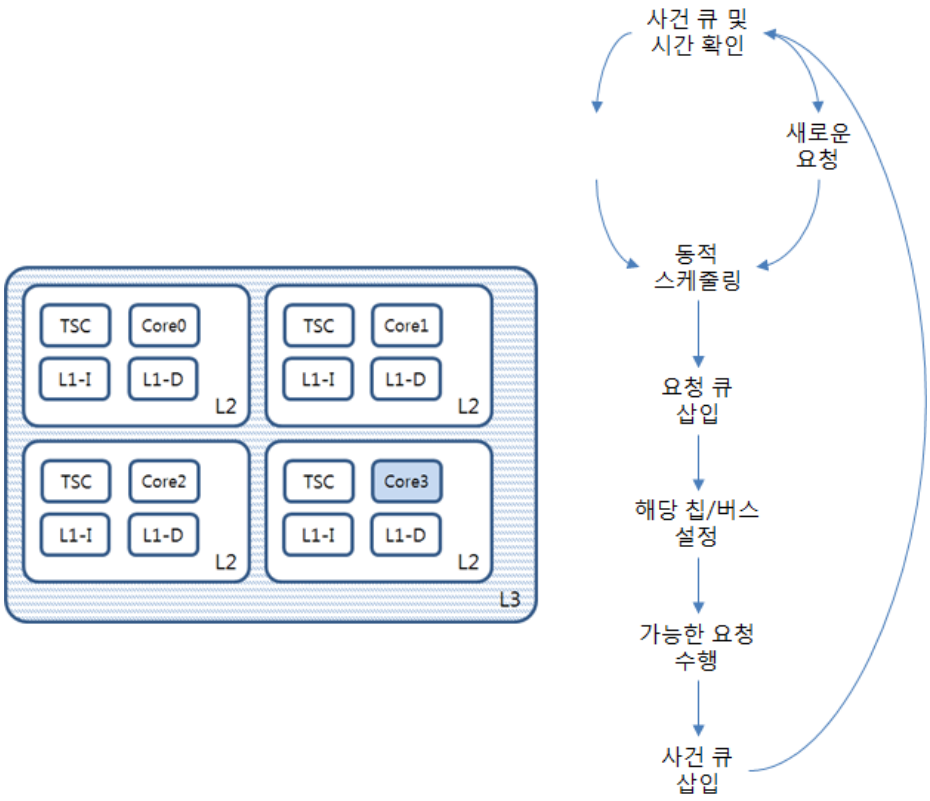

그림 7. 시뮬레이션 수행 프로세스와 전담 프로세서 코어

간격으로 값이 증가하는 특수목적 레지스터인 타임스탬프카운터가 벽시계로 활용할 수 있도록 제공되고 있다. 본 시뮬레이션에서는 이산사건의 시작을 기준으로 정해진 시간 만큼의 흐름을 파악할 수 있도록 성능카운터 레지스터 값의 증가속도를 고려하여 예상하는 값보다 크거나 같아지는 종료시점까지 성능카운터를 반복하여 확인하는 방법으로 시간흐름을 처리하였다.

성능카운터를 반복하여 확인하는 작업과 시뮬레이션 작업 처리가 스케줄링으로 인하여 서로 시간적 영향을 주는 상황을 예상할 수 있다. 이로 인하여 시간확인의 정밀성과 시뮬레이션의 시간적

정확성이 영향을 받지 않도록 성능카운터를 반복 확인하는 작업은 메모리 전송 작업과 분리하여 그림 7 과 같이 해당 프로세스에 별도의 프로세서 코어를 할당하였다.

■ 인과적 후속 사건의 처리

발생한 이산사건이 시간 지연후 종료되면 시뮬레이션의 상태가 바뀌고 사건의 발생으로 인한 인과율에 따라 후속사건이 발생하게 된다. 새로 발생하는 후속사건은 다시 사건 큐로 삽입되어 동적스케줄러의 선택을 기다리게 된다.

플래시 메모리 제어기의 동작과정에서 발생하는 시간흐름과 관련있는 모든 사건은 사건 큐에 입력되어 대기하고 출력되어 처리된다. 사건으로 인한 지연시간의 대부분은 플래시 칩의 동작 지연시간으로부터 기인한다. 처리된 사건으로 말미암아 인과적으로 발생하게 될 새로운 사건들은 다시 사건 큐에 입력되어 처리를 기다린다. 읽기 동작의 경우 칩 내부 동작 후 데이터 전송 동작을 기다리게 되고 쓰기 동작의 경우 데이터 전송 동작 후 칩 내부 동작을 기다리게 된다.

4.3.2 하드웨어 모델링

하드웨어 모델링 모듈집합은 무순서 플래시 메모리 제어기의 실제 기능적 동작을 모사한다. 각각의 모듈은 스레드로 분리되어 동작하고 모듈간에는 메모리 공간을 공유하여 통신한다. 본 절에서는 4.2.2 절에서 소개한 하드웨어 모델링 모듈집합의 상호작용을 설명하여 제어기 동작의 모사를 설명한다.

■ 작업 큐와 사건 큐의 동기화

사건 큐는 전체 플래시 메모리 제어기 시뮬레이션의 사건들을 담고있다. 시뮬레이션의 사건들은 플래시 동작의 개시와 종료, 임의의 시간에 비동기적으로 발생하는 외부오류와 같은 제어기 내외부의 의미 있는 상태변화를 말한다. 사건 큐로 입력되는 사건은 FTL 에서 제어기로 입력되는 패킷형태의 요청, 비동기오류발생기에서 발생시킨 오류가 있다. 시뮬레이션에서 일어나는 일련의 사건들은 사건 큐에서 유지되고 이들 중 제어기 요청은 작업 큐와 사건 큐의 동기화를 통하여 작업 큐로 이동한다.

■ 작업 큐의 요청과 플래시 동작수행유닛의 응답

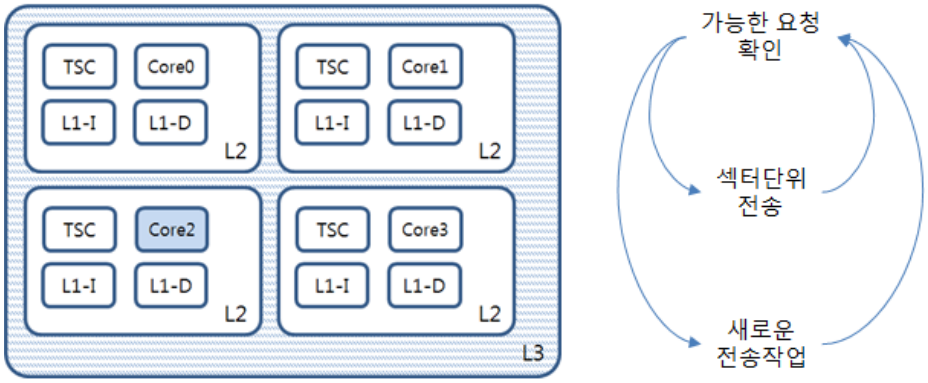

그림 8. 데이터 전송 프로세스와 전담 프로세서 코어

작업 큐는 시뮬레이션의 사건들 중 플래시 메모리 제어기에 입력된 요청들을 담고있다. 이 요청들은 동적스케줄러의 선택에 의해 처리된다. 동적스케줄러가 선택한 요청은 플래시 동작수행유닛으로 전달되고 동작수행유닛은 플래시 메모리 모듈과 해당 요청과 관련된 데이터 전송작업을 수행한다.

데이터 전송작업은 시뮬레이션의 진행과 독립적으로 진행할 수 있도록 그림 8 과 같이 데이터 전송을 위한 별도의 프로세서 코어를 할당하였다. 할당된 데이터 전송용 프로세서 코어는 플래시 요청에 따르는 데이터 전송을 전송큐의 우선순위에 따라 섹터단위로 수행하고 모두 전송하면 다음 우선순위의 데이터를 전송한다. 데이터 전송 큐의 자료구조는 페이지와 버퍼를 모사하는 메모리 주소와 섹터단위의 남은 전송량을 가지고 우선순위에 따라 큐 내부에 정렬한다.

■ 플래시 동작수행유닛의 플래시 상태 변경

플래시 동작수행유닛은 동적스케줄러에 의해 작업큐에서 전달받은 요청에 따라 플래시 메모리 모듈과 데이터 전송작업을 수행하고 그에 따른 플래시 칩의 요청 큐, 칩과 버스의 상태를 갱신한다.

■ 동적스케줄러의 작업 큐 요청 선택

동적스케줄러는 스케줄링 정책에 따라 작업 큐 내부의 요청들 가운데 처리할 요청을 선택한다. 스케줄러는 칩과 버스의 상태를 관찰하면서 수행이 가능한 요청들을 식별하고 가능한 요청들의 대상 칩들 가운데서는 라운드 로빈 방식에 따라 순서대로 요청 처리를 할당한다.

4.3.3 오류 발생기

오류 발생기 모듈 집합은 플래시 메모리 동작 중 발생하는 오류들을 모사한다. 플래시 메모리에서 발생하는 오류와 그로 인한 플래시 메모리의 상태를 분류한 [7]에 따라 본 시뮬레이션 에서는 비동기적 오류와 동기적 오류를 발생시키는 모듈을 분리하여 구현하였다.

■ 비동기적 오류 발생기와 오류의 처리

비동기적 오류는 플래시 메모리의 내부 동작과 관계 없이 임의의 시간에 외부로부터 발생하는 오류이다. 이와 같은 오류로 전원 오류를 들 수 있다. 전원 오류는 플래시 내부 동작과 별도의 인과 상황으로 인하여 발생하기 때문에 해당 오류는 시뮬레이션 전체의 사건으로서 사건 큐에 입력되어 처리된다. 사건 큐에 입력된 전원 오류는 오류의 해당 시간에 발생이 모사되어 현재 제어기 내부에 입력되어 외부로 응답하지 않은 처리중인 요청들을 파악한다.

■ 동기적 오류 발생기와 오류의 처리

동기적 오류는 플래시 메모리의 내부 동작시 소자의 물리적 특성에 따라 동작시점에 발생하는 오류이다. 이와 같은 오류로는 비트 반전 오류와 같은 일시적 오류, 셀의 내용이 0 또는 1 로

고정되는 영구적 오류를 들 수 있다. 이들 동기적 오류는 플래시 내부 동작이 오류의 인과 상황으로 작용하기 때문에 해당 오류는 작업 큐에서 플래시 동작이 수행되는 시점에 입력되어 처리된다. 플래시 동작의 수행시점에 발생한 동기적 오류는 오류 내용에 따라 수행유닛이 플래시 메모리 모듈, 칩과 버스 상태에 반영하여 처리한다.

4.4 실시간 시뮬레이션의 정확성과 예측가능성

실시간 시뮬레이션의 정확성을 보이기 위해서는 입력 작업부하의 시간 지연을 모사 하는 시뮬레이션 내부 작업의 스케줄링이 가능함을 보여야 한다. 본 절에서는 시뮬레이션이 실시간으로 동작할 수 있음을 보이기 위하여 시간 파악의 정확성 및 시뮬레이션의 예측가능성과 관련된 이슈들을 파악하고 대책에 대하여 논의한다.

4.4.1 시간 파악의 정확성

본 시뮬레이션에서는 시간흐름 확인하기 위하여 벽시계로 프로세서의 타임스탬프카운터 레지스터를 확인한다. 따라서 시간 지연을 정확하게 모사하기 위해서는 시간 흐름의 기준이 되는 벽시계를 확인하는 작업의 빈도가 중요하다. 레지스터를 충분한 빈도로 확인하지 못하면 긴 확인 주기로 인해 시간 파악의 오차가 생길 수 있고 누적되어 시뮬레이션의 시간적 정확성에 문제를 일으킬 수 있기 때문이다.

시간 흐름을 파악하는 프로세스에서 레지스터를 확인하는 빈도에 영향을 미치는 요소들을 다음과 같이 분류하였다.

■ 운영체제의 스케줄링

실시간으로 동작하는 시뮬레이션에서 현재 시간을 파악하는 작업의 주기에 따라 시간 처리의 정확성이 결정된다. 어떠한 실시간 작업이 제한 시간안에 처리되어도 현재 시간을 확인하는 작업이 늦어지면 후속 작업 시점의 정확성에 영향을 미칠 수 있기 때문이다. 시간파악의 정확성에 영향을 미치는 첫번째 요소는 운영체제의 스케줄링으로 인한 시간 파악의 지연이다.

현대의 운영체제 내부 작업단위인 프로세스는 프로세서를 무한하게 사용할 수 없다. 운영체제의 관리하에 있는 많은 프로세스가 시간을 분할하여 프로세서를 공유하기 때문이다. 프로세스는 운영체제로 부터 일정한 시간을 할당 받아야 프로세서를 사용할 수 있고 할당 받은 시간이 만료되면 운영체제가 프로세서를 회수하거나 자신의 요청에 의해 프로세서를 운영체제에게 반환하게 된다. 프로세서를 돌려 받은 운영체제는 자신의 인스트럭션을 실행할 준비를 하고 기다리고있는 준비 목록의 프로세스들 가운데 하나를 선택하여 프로세서를 할당한다. 프로세서를 프로세서할당받은 프로세스는 다시 운영체제에게 프로세서를 돌려 줄 때까지 자신의 코드를 실행한다. 이것이 운영체제의 전형적인 프로세서 스케줄링이다.

프로세스가 운영제체에 의해 프로세서를 회수 당하고 다시 스케줄링 될 때 까지의 지연시간은 밀리초 단위까지 길어질 수 있다. 마이크로초 단위의 실시간 처리를 보장해야 하는 플래시

메모리 제어기 시뮬레이션에서 시간 확인작업을 위해 프로세서 코어를 할당받은 프로세스가 운영체제의 스케줄링에 따라 프로세서를 회수당하지 않도록 해야한다.

본 시뮬레이션에서는 이를 위해 리눅스 부팅 설정시 시간 확인작업을 전담할 프로세서 코어를 운영체제가 시간을 분할하여 스케줄링 하지 못하도록 코어를 스케줄링에서 고립시켰다. 고립된 코어는 수행시 해당 코어를 지정한 프로세스만이 사용할 수 있고 스스로 작업을 마치고 코어를 내놓기 전까지는 운영체제에 의해 선점되지 않는다. 이와 같이 고립시킨 코어에서 시간을 확인하는 간격은 96% 이상 10 나노초 이하를 보장한다.

■ 코어들 간의 계수 오차

시뮬레이션의 실행 환경인 인텔의 제온 E5 1620 프로세서는 총 4 개의 코어를 제공한다. 4 개의 코어들은 시간에 따라 값이 증가하는 타임스탬프카운터(Time Stamp Counter) 레지스터를 비롯한 물리적 상태 레지스터를 각각 갖고있다. 각 코어들이 상호 독립적으로 동작하기 때문에 작용하는 물리적 환경에 차이가 생길 수 있다. 현재 인텔에서는 코어 간 타임스탬프카운터 레지스터의 계수 오차가 없다고 보장하지 않고 있다. 따라서 2 개 이상의 코어를 사용하는 작업에서 타임스탬프카운터 레지스터의 값을 이용하는 경우 각 레지스터간의 값들을 일정시간마다 동기화하거나

일정하게 같은 코어에서 값을 확인한다는 것을 보장해야 한다. 본 시뮬레이션에서는 시간을 확인하는 프로세스를 담당하는 코어를 고립시킨 코어에 배타적으로 할당하였다.

■ 다중 논리적 프로세서

레지스터와 메모리의 값으로 규정되는 프로세스의 상태는 운영체제 스케줄링시 레지스터의 값이 프로세스제어블록에 저장되는 과정을 거쳐. 문맥전환이 일어난다. 이는 프로세스 수준의 문맥전환으로 운영체제의 판단을 통해 전환시점이 결정된다.

이와 별개로 현대의 프로세서들은 하나의 코어에 상태 레지스터들의 집합을 별개로 구성하고 인스트럭션수행중 비는 시간동안 하드웨어 수준에서 문맥을 전환하는 다중 논리적 프로세서 기술을 사용한다. 따라서 하나의 프로세스를 운영체제가 물리적 프로세서에 전담을 시켜도 다른 논리적 프로세서가 하드웨어 수준에서 문맥전환이 될 수 있다.

본 시뮬레이션에서는 이러한 다중 논리적 프로세서를 비활성화 시키고 시간 확인 작업을 하는 프로세스와 작업을 전담할 프로세서가 일대일로 연결되어 수행될 수 있도록 하였다.

■ 시뮬레이션의 수행 시간

그림 9. 시뮬레이션 수행 과정

시뮬레이션 수행은 시간을 반복하여 확인하다가 사건 큐에 시간이 만료된 사건에 발견될 경우 해당 사건을 처리하는 과정을 반복하여 진행한다. 따라서 시뮬레이션의 정확성을 위해서는 시간의 반복확인 작업과 사건 큐의 처리작업에 걸리는 시간을 측정하고 해당 시간을 시뮬레이션에서 고려해야 한다. 수행시간으로 인해 시뮬레이션의 시간확인 간격이 길어져 정확성에 영향을 미칠 수 있기 때문이다.

그림 9 의 시뮬레이션 수행에서 사건 큐와 시간을 반복하여 확인하는 작업은 실험환경에서 10ns 이하로 관측되었고 후속사건

처리와 새로운 요청의 처리는 각각 최대 700ns 와 150ns 가 관측되었다. 새로운 스케줄링에 의해 요청 큐에 요청이 삽입되는 과정이 최대 2us 로 관측되었고 요청을 수행하기 위한 칩과 버스 설정, 실행 가능한 요청을 수행, 그리고 사건 큐에 해당 사건을 넣는 과정은 각각 100us, 120ns, 120ns 로 관측되었다. 이 경우 사건 큐의 사건을 처리하기 위한 수행시간을 해당 수행시간의 합으로 한정하고 사건 큐에서 해당 수행시간 이내에 만료되는 사건들을 미리 완료 처리하여 정확성을 확보할 수 있다.

4.4.2 동작의 예측가능성

본 시뮬레이션은 무순서 플래시 메모리 제어기의 실시간 동작을 모사하기 위하여 시간확인, 사건 큐와 작업 큐의 조작, 데이터 전송, 이벤트 선택 등의 내부 동작을 수행한다. 전체 시뮬레이션의 실시간 동작을 보장하기 위해서는 시뮬레이션을 이루는 부분적 동작들이 예측가능한 범위내에서 마침을 보여야 한다. 이를 위하여 부분적 동작의 예측가능성에 영향을 미치는 요소들을 다음과 같이 분류하였다.

■ 운영체제의 스케줄링

현대의 범용 운영체제는 여러 프로그램을 실행시켜 메모리에 올려놓고 프로세서를 공유하는 다중 프로그래밍을 지원한다. 운영체제는 메모리 상에서 실행 대기중인 준비 큐의 프로세스 중 하나를 선택하여 일정 시간 프로세서를 사용하도록 할당하고 시간이 만료되면 다른 프로세스를 선택하여 프로세서를 할당하는 스케줄링 과정을 반복한다.

할당할 프로세스를 선택하는 기준은 운영체제의 정책에 따라 달라지는데 각각의 프로세스 입장에서 운영체제의 스케줄링은 벽시계 기준의 프로그램 실행시간을 예측할 수 없게 된다. 프로세서를 할당 받은 후 다음 프로세서를 할당 받을 때 까지의 지연시간은 시스템 내 다른 프로세스들의 수와 정책에 의해 영향을 받기 때문이다.

따라서 실시간 시뮬레이션을 실행하기 위한 환경에서는 예측가능성을 위하여 운영체제의 스케줄링으로 인한 영향을 제거해야 한다. 이를 위하여 본 시뮬레이션 환경에서는 실시간 시뮬레이션을 실행시킬 프로세서 코어를 운영체제의 범용 스케줄링 정책에서 제외 시켜 지정한 프로세스 만이 실행될 수 있도록 하였다.

■ 공유자원에 대한 접근 제어

여러 구성요소를 가진 하드웨어에서 각 요소는 전원이나 동력이 공급되는 경우 동시에 동작하고 요소들 간의 인터페이스에 따라 상호작용한다. 그러나 이러한 하드웨어를 시뮬레이션하는 소프트웨어는 동시성에 제한이 있어 요소들 간의 상호작용에 따른 정보를 자료구조를 통해 공유해야한다. 시뮬레이션을 구성하는 프로세스가 다수인 경우 동작에 필요한 자료구조를 공유할 때 적절한 동기화 매커니즘을 적용하지 않으면 경쟁 조건으로 인한 오류가 발생할 수 있다.

하드웨어의 구성요소를 시뮬레이션의 독립적인 프로세스로 구성할 때 경쟁으로 인한 성능저하를 고려해야 한다. 시뮬레이션에서 하드웨어의 구성요소에 따라 별도의 프로세스를 구성하면 공유하는 정보를 담고있는 자료구조들에 대한 경쟁과 프로세스의 잦은 잠듬(Sleep)과 깨움(Wake) 으로 인한 오버헤드가 심해지기 때문이다.

본 시뮬레이션에서는 하드웨어 동작을 모사할 때 구성요소 들을 각각 동작하는 프로세스로 구성하는 대신 각 구성 요소들이 동작하는 시점 시뮬레이션과 실제 데이터처리의 두 프로세스로 구분하여 공유 자료구조의 경쟁조건과 프로세스 관리로 인한 오버헤드를 완화하였다.

■ 데이터 전송의 유효대역폭과 스케줄링

시점 시뮬레이션과 데이터 전송을 분리하여 시뮬레이션을 구성하면 데이터 전송량 만큼의 시뮬레이션 대상 플래시 메모리 제어기의 설정이 가능하다. 실시간으로 동작하는 시뮬레이션에서 하나의 요청으로 인하여 발생하는 데이터 전송 요구는 요청이 끝나는 실제 시간의 마감시한 이전에만 처리되면 되기 때문이다.

데이터 전송의 대역폭은 제어기 시뮬레이션의 플래시 메모리 설정의 제한으로 작용한다. 플래시 메모리 칩의 데이터 전송과정을 시뮬레이션 하는 과정에서 처리 할 수 있는 데이터 전송의 한계가 있기 때문이다.

고정된 마감시한을 기준으로하는 요청들의 스케줄링은 마감시한이 빠른 요청들을 먼저 처리하는 최단 마감 우선 스케줄링(Earliest Deadline First)기법이 최적이다[8]. 따라서 본 시뮬레이션에서도 최단 마감 시간을 가진 요청들 부터 처리하므로써 스케줄링의 최적성을 만족시키는 것을 목표로 삼았다.

하지만 플래시 메모리 제어기로 전달되는 요청들은 고정된 마감시한을 제공하지 못한다. 플래시 메모리 칩들 간의 버스 공유상태에 따라 읽기 요청의 데이터 전송 시점, 쓰기 요청의 상태 확인 시점이 지연되어 마감시간이 바뀔 수 있기 때문이다. 더욱이 요청의 스트림은 시뮬레이션이 진행되면서 언제든지 새로 들어올 수 있다. 새로 들어온 요청들에 따라 최단 마감 우선 스케줄링을 하려면 이전 요청들까지 고려하여 새로 스케줄링을 해야한다.

$$\frac{T_{Read}}{\#\ of\ Chip \times t_{sRead}}$$

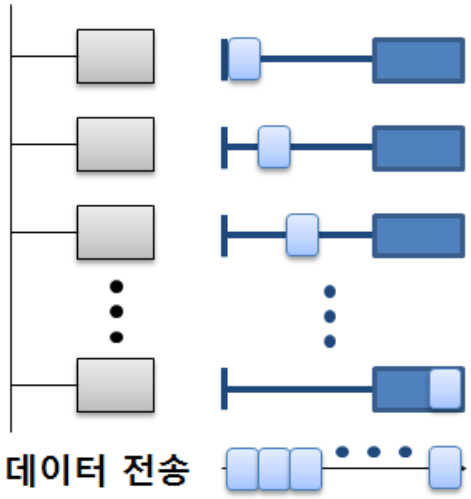

그림 10. 버스 수 증가에 따른 전송량 제한

요청들을 스케줄링하는 시점에 정확한 마감시간을 알 수 없는 상황에서 문제를 접근해야 하는 상황에서 근사적 스케줄링을 구현하고 평가하였다. 구현한 근사적 접근은 버스에 대한 경쟁을 고려하지 않고 요청의 전송시점에서 해당 요청의 마감시간을 기준으로 데이터 전송을 스케줄링 하여 처리하는 방식이다.

구현한 시뮬레이션에서 데이터 전송 시간은 512 바이트를 하나의 섹터로 가정하였을 때 평균 50ns 가 지연되었다. 캐쉬의 수준에 따라 지연시간의 차이가 있었지만 80%이상의 전송이 50ns 안에서 일어났고 시뮬레이션 수행 프로세스와 데이터 전송 프로세스를 담당하는 두 프로세서 코어가 L3 캐쉬를 공유하는 환경에서 지연시간의 오차는 크지 않았다.

해당 지연시간을 기준으로 데이터 전송량을 초과하지 않는 지원 설정의 제한을 계산하면 아래와 같다. 마감 시간이 가장 빠른 읽기

$$\frac{T_{Read} + \#\ of\ Chip \times t_{dRead}}{Inc.\#\ of\ Chip\ \times \#\ of\ bus\ \times t_{sRead}}$$

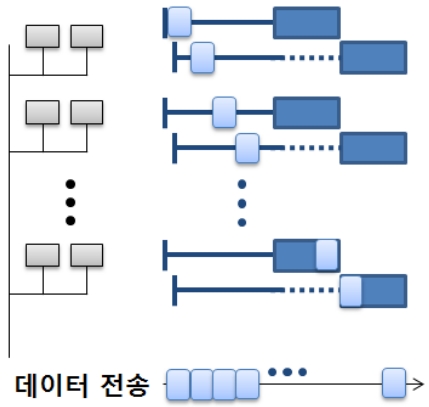

그림 11. 칩 개수 증가에 따른 전송량 제한

요청을 기준으로 칩의 개수 증가와 버스 수 증가를 상황을 나누어 계산하였다.

그림 10 과 같이 버스가 증가하는 경우 모든 버스에 읽기 요청이 스케줄되었을 때 데이터 전송의 최악의 경우가 발생하게 된다. 이러한 경우 실제 읽기 시간 T_{Read} 가 마감시한으로 각 버스의 칩 수 만큼의 데이터 전송시간 t_{sRead} 이 걸리게 된다.

그림 11 과 같이 칩의 수가 증가하는 경우 모든 칩에 읽기 요청이 스케줄되었을 때 데이터 전송의 최악의 경우가 발생하게 된다. 이러한 경우 실제 읽기 시간인 T_{Read} 에 버스를 공유하는 칩이 버스에 대한 경쟁으로 인하여 마감시간이 t_{dRead} 만큼 연장된다. 늦게 스케줄링되는 읽기 요청을 처리하는 칩의 마감시간은 버스를 공유하는 칩의 수 만큼 증가한다. 각 버스에서 칩의 수가 증가하게 되면 버스의 수를 곱한 만큼의 t_{sRead} 데이터 전송시간이 걸리게 된다.

5 장에서는 위의 근사 스케줄링 방법에 대하여 최적인 실제 마감 시간 우선처리 기법과의 유사성을 실험을 통하여 평가하였다

.

제 5 장 실험 및 평가

무순서 플래시 메모리 제어기 시뮬레이션을 구현하여 두가지 실험을 수행하고 결과를 평가하였다. 첫째, 근사 스케줄링과 최단 마감시간 스케줄링의 유사도를 평가하는 실험으로 근사 스케줄링한 데이터 전송 처리순서와 요청 처리후 얻어진 실제 마감시간의 순서를 비교하였다. 둘째, FPGA 로 구현된 실제 무순서 플래시 메모리 제어기 하드웨어와의 유사도를 평가하는 실험으로 동일하게 설정된 시뮬레이션과 하드웨어에 동일한 작업부하를 입력하고 단계별 처리 시간과 전체응답시간을 비교하였다. 본 장에서는 위의 실험환경을 소개하고 각 실험과정 및 결과에 대하여 설명한다.

5.1 실험 환경

5.1.1 시뮬레이션 동작 환경

시뮬레이션은 인텔 기반 다중코어 프로세서가 장착된 PC 환경에서 리눅스 커널 기반 운영체제를 통하여 동작한다.

PC 환경에서는 성능의 일관성을 위하여 동적 전압 변화 기능을 비활성화 하였고, 인스트럭션 수준에서의 예측가능성을 위하여 다중 논리적 프로세서 기능을 비활성화하여 하나의 프로세서 코어당 하나의 상태 레지스터 집합이 사상되도록 하였다.

운영체제 환경에서는 시뮬레이션을 전담하여 수행할 프로세서 코어를 운영체제의 스케줄링으로 부터 고립시키고 특정 프로세스가 지정한 코어에서만 수행되도록 리눅스 커널의 프로세서 스케줄링 마스크 관련 코드 일부를 수정하였다.

5.1.2 하드웨어 동작 환경

실제 동작 시간을 측정하고 시뮬레이션과 비교하기 위하여 플래시 메모리 개발 플랫폼 상에 FPGA 로 구현된 무순서 플래시 메모리 제어기의 하드웨어를 이용하였다.

해당 보드의 Spartan6 FPGA (XC6SLX150T) 에 Microblaze 를 구현하여 사용하였다. 플래시 메모리 모듈은 2 개의 NVRAM 용 슬롯에 Toshiba 의 NAND 플래시 메모리를 탑재하여 사용하였다. 각 슬롯은 2 개의 8 비트 폭의 플래시 버스를 지원한다.

해당 하드웨어 동작환경에서 4.3.1 절에서 설명한 플래시 메모리 칩의 동작시간을 측정하였고 시뮬레이션과 비교하기 위하여 동일 워크로드에 대한 응답시간을 측정하였다.

5.2 근사 스케줄링 평가

본 시뮬레이션은 요청을 처리할 때 요청에 따른 플래시 동작의 실시간 재현과 해당 데이터의 전송을 독립적으로 처리하는

방식으로 구현되었다. 데이터 전송은 요청이 시뮬레이션의 사건 큐에 삽입되어 해당 요청에 따른 칩동작과 버스 점유 시간이 시뮬레이션 상에서 흐르고 마감시간 이전까지 처리되어야 한다. 해당 상황에서 최적의 접근 방법인 최단 마감시간 스케줄링을 구현할 때 실제 요청들 간 버스 경쟁과 새로운 요청의 스케줄링으로 인한 마감시간의 변동이 생길 수 있다.

이러한 불확실성을 감안하여 본 시뮬레이션에서는 버스 경쟁에 대한 고려를 배제한 근사 스케줄링을 제안하였다. 근사 스케줄링은 실제 플래시 요청 처리에서 버스 경쟁에 의한 요청순서 변동은 많지 않다는 점과, 칩 동작시간은 길어지고 데이터 전송시간은 짧아지는 플래시 메모리의 기술적 흐름을 반영한 접근 방법이다. 위의 근사 스케줄링을 평가하기 위하여 최적 접근인 최단 마감시간 스케줄링과 결과의 차이를 분석하는 실험을 수행하였다.

실험은 과정은 다음과 같다. 버스 4 개, 칩 4 개의 설정에서

단위: %

버스 \ 칩	4	8	16	32
4	**89**	**90**	**91**	**91**
8	**92**	**90**	**91**	**91**
16	**89**	**90**	**91**	**91**
32	**90**	**90**	**91**	**91**

표 1. 근사 스케줄링과 최단 마감시간 스케줄링의 유사도

시작하여 버스와 칩의 개수를 증가시키면서 3000 개의 요청을 입력하였다. 근사 스케줄링에 의해 요청의 데이터 전송이 처리된 순서와 모든 요청의 실시간 처리가 끝난 후 요청의 실제 최단 마감시간 순서를 비교하였다. 결과는 표 1 과 같다.

비교 결과 실제 최단 마감시간 기준의 요청순서와 버스 경쟁을 배제하고 마감시간을 가정한 스케줄링의 요청순서는 약 90% 일치하였다.

5.3 하드웨어 응답시간 비교

무순서 플래시 메모리 제어기가 FPGA 형태로 구현되어 있는 개발 보드상에서 해당 하드웨어의 응답시간과 시뮬레이션의 응답시간을 비교하였다.

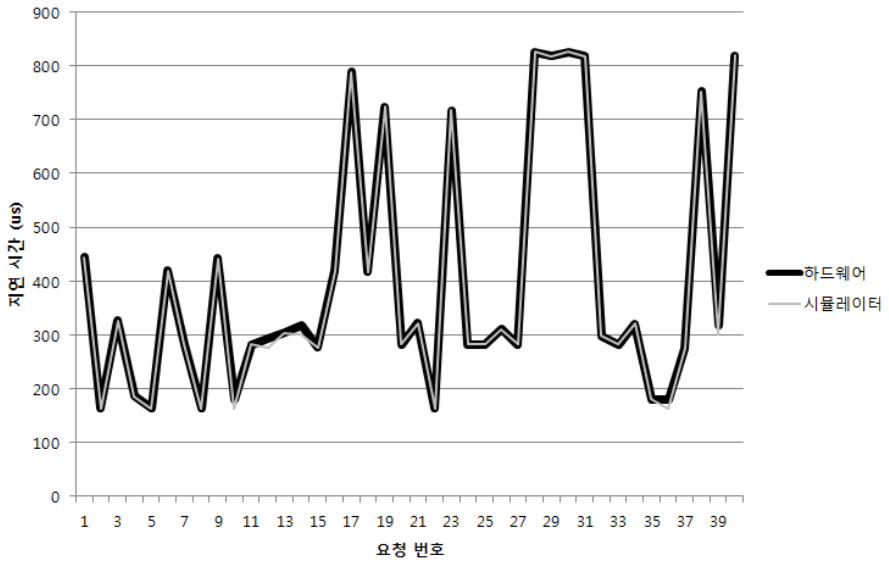

그림 12. 하드웨어와 시뮬레이션의 응답시간 비교

위 그림 12 에서 버스를 공유하는 두 개의 칩에 대하여 40 개의 요청을 보내고 하드웨어의 응답시간과 시뮬레이션의 응답시간을 비교하였다. 응답시간의 오차를 계산하였을 때 99.6%의 유사성을 보였다.

제 6 장 결론

본 논문에서는 플래시 메모리 기반 저장장치의 성능과 신뢰성을 일반 PC 환경에서 평가하기 위하여 무순서 플래시 메모리 제어기의 시뮬레이션을 구현하였고 이를 실제 하드웨어와 비교 평가하였다. 또한 시뮬레이션 동작환경의 성능을 바탕으로 시간적 정확성과 결과의 예측가능성을 보장할 수 있는 저장장치 설정의 상한을 제시하였다.

본 시뮬레이션을 통하여 얻을 수 있는 이점은 다음과 같다.

1. 플래시 저장장치의 개발 시 하드웨어 개발과 소프트웨어 개발을 독립적으로 수행할 수 있어 개발의 효율을 높일 수 있다.
2. 플래시 저장장치의 내부설정과 소프트웨어 계층인 FTL 의 동작에 따른 실시간 성능을 측정 가능하다.
3. 플래시 저장장치의 동기적 내부오류와 비동기적 외부오류를 원하는 시점에 발생시켜 FTL 의 신뢰성을 쉽게 평가할 수 있다.

시뮬레이션의 실시간 성능과 정확성을 평가하기 위하여 리눅스 환경에서 운영체제의 스케줄링에서 독립시킨 프로세서 코어를 시뮬레이션을 수행에 전담시켜 구현하였다.

정확성을 평가하기 위하여 실제 하드웨어와의 비교 결과 동일한 내부 설정 상태와 작업부하에 대하여 하드웨어의 처리 시간과 시뮬레이션의 처리시간이 99.6% 동일하였다.

예측가능성을 평가하기 위하여 플래시 요청 처리에 따른 시뮬레이션의 데이터전송 순서를 최단마감시간우선 (Earliest Deadline First) 스케줄링을 목표로 하여 근사적으로 접근하였고 모든 요청을 처리한 후 실제 마감시간과 순서를 비교한 결과 실제 플래시 요청의 마감시간과 약 90% 일치하였다.

본 시뮬레이터와 관련하여 다음과 같은 향후 연구가 필요하다.

예측가능성을 높이기 위하여 근사 최단마감시간우선 스케줄링의 정확도를 높이는 것이 필요하다. 현재의 정확도 내에서 상용 플래시 메모리 저장장치의 설정을 지원하기에는 충분하지만 향후 칩과 버스의 수가 늘어나는 경우를 대비하여 시뮬레이터의 계산부담을 높이지 않으면서 스케줄링의 정확도를 높일 필요가 있다.

신뢰성 향상을 위하여 본 시뮬레이터를 이용한 FTL 의 개발 시 FTL 이 제대로 처리하지 못한 오류가 발생하였을 때 이전 동작들에 대하여 인과적 추적이 가능하도록 지원할 필요가 있다. 단순한 오류 발생 여부가 아닌 해당 오류에 대하여 페이지 동작모델에 의한 FTL 의 동작 이력을 개발자에게 제시할 수 있다면 오류의 근본원인을 제거하는데 더욱 도움이 될 것이다.

위와같은 예측가능성 개선과 사용자 편의성 확장을 통하여 본 시뮬레이터를 이용한 플래시 저장장치 개발 시 효율과 신뢰성을 높일 수 있을 것으로 기대된다.

참고 문헌

[1] J. Bucy, G. Ganger, and et al., “The DiskSim Simulation Environment Version 3.0 Reference Manual,” http://citeseer.ist.psu.edu/bucy03disksim.html.

[2] J. Griffin, J. Schindler, S. Schlosser, J. Bucy, and G. Ganger., “Timing-accurate Storage Emulation,” In Proceedings of the First USENIX Conference on File and Storage Technologies (FAST’02), pages 75–88, Monterey, CA, January 2002.

[3] Prabhakaran, V. and Wobber, T., "SSD Extension for DiskSim Simulation Environment," http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/, 2009

[4] M. Jung, ET AL. NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level. In International Conference on Massive Storage Systems and Technology (MSST), 2012.

[5] J. Yoo, Y. Won, J. Hwang, S. Kang, J. Choi, S. Yoon, and J. Cha, “Vssim: Virtual machine based ssd simulator,” in Mass

Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on, 2013, pp. 1–14.

[6] E. Nam, B. Kim, H. Eom, and S. Min, “Ozone (O3): an out-oforder flash memory controller architecture,” IEEE Trans. Computers, vol. 60, no. 5, pp. 653-666, May 2011.

[7] J. Yun, J. Yoon, E. Nam, and S. Min, “An Abstract Fault Model for NAND Flash Memory,” IEEE Embedded Systems Letters, vol. 4, no. 4, pp. 86-89, 2012.

[8] J. Liu. Real-time Systems, Upper Saddle River, NJ: PEARSON, 2000, pp. 67.

Abstract

High-fidelity and Real-time Simulation of Out-of-Order Flash Memory Controller for Performance and Reliability Testing of Flash-based Storage Device

Jongbo Bae

Department of Electrical Engineering and Computer Sciences

The Graduate School

Seoul National University

In the development of flash memory based storage device, the design of storage hardware is carried out in parallel with the internal software. In developing the internal software, difficulty in testing the reliability or performance changes of software with hardware prototype necessitates the software testing environment.

This dissertation presents, a simulation of out-of-order flash memory controller which shows accurate timing-latency and detail error behavior. Out-of-order (Ozone) flash memory controller improves performance through issuing multiple flash

operations regardless of arrival ordering. The simulation of Ozone flash memory controller was designed an implemented as an event-driven simulation, by analyzing the hardware O3 controller. In order to get the accurate timing of the simulation, we obtained flash memory operation latency from commodity NAND flash chip and used it to simulate the latency. Moreover, to obtain the similarity of fault environment, we simulate the external power fault asynchronous with flash operation and internal flash errors synchronous with flash operation based on the fault model which classifies possible internal / external fault. Through timing accuracy and fault similarity of the simulation, development of flash memory based storage device can be more efficient by evaluating the performance and testing the reliability on commodity PC environment.

In order to validate the simulation of Ozone (O3) flash memory controller, we compared the performance of the simulation and the prototype implemented on the FPGA. With the same settings and the workloads, result showed that performance of the prototype and the simulation was 99.6% identical.

keywords : Flash Memory-based Stroage Device, Flash Memory Controller, Flash Translation Layer (FTL), Real-time Simulation

student number : 2013-20797